CS 166 Homework 1

Leo Martel (lmartel)

4/9/2014

Question 1 (Sparse Tables with O(1) Queries)**.** *Design a data structure to compute largest k for* $2^k \leq j-i+1$ *in constant time for use in RMQ.*

Overview: we simply precompute k such that $2^k \leq i < 2^{k+1}$ for $1 \leq i \leq n$ and store the values in an array.

Representation: an array of integers.

Preprocessing: Initialize an array A of length n+1 (allowing one-indexing for simplicity).

Starting with $k = 0, p = 2^k = 1, x = 1$. Repeat ([If $2p \leq x$ then increment k and set $p = 2p$.] Write $A[x] = k$. Increment x.) until the array is filled in, and $x = n + 1$.

Each step gives us the correct k, because we start with correct $k = 0$ for $x = 1$, and by induction if k is correct for $x - 1$, either $k + 1$ is correct for x (which we check) or k is correct for x, which we fall back to. This is because adding 1 never moves us past more than one power of two.

Each step takes $O(1)$ time, doing only multiplication, an array write and a few increments. We do n steps, since we increment x each time from 1 to n. So the total preprocessing work is $O(n)$

Querying: $A[j - i + 1] = k$ gives us the largest k such that $2^k \leq j - i + 1$, as shown in the previous step.

The minimum range $j - i + 1 = 1$ and the maximum range is n, so all possible valid queries can be answered by the array.

A query requires some arithmetic and an array access, which takes $O(1)$ time.

Question 2 (Area Minimum Queries)**.**

i. An $\langle O(mn), O(min(m, n))\rangle$ data structure for AMQ.

Note: assume WLOG that $m < n$. That is, we have more columns than we have rows. If not, rotate your screen or switch all the m's and n's in this proof.

Overview: we solve RMQ for each row of the rectangle, then iterate through our RMQs on each AMQ query.

Representation: we use an array of m Fischer-Heun structures, each of size n. The Fischer-Heun structures are represented with arrays (sparse tables on top and precomputed cartesian-tree-indexed RMQs on the bottom)

Preprocessing: build a Fisher-Heun Structure on each row of A. So, $(i, 0), (i, n-1)$ for $0 \leq i < n$. This will build m structures of size n, so the total preprocessing time is $m * O(n) = O(mn)$, since Fisher-Heun Structures were shown in lecture to take linear preprocessing time.

Querying: To answer $AMQ_A((i, j), (k, l))$:

Use the Fisher-Heun Structures to find $RMQ_x(j, l)$ for rows $x \in \{i, i + 1, ..., k\}$ and take the running minimum of those results. We will need at most m queries, since we have m rows. Each query takes $O(1)$ time as shown in lecture, so our total query time is $O(m) = O(min(m, n))$ by the **note**.

ii. An $\langle O(mn \lg m \lg n), O(1)\rangle$ data structure for AMQ.

Overview: we build a sparse table of rectangles with dimensions 2^i by 2^j. We can construct any queried rectangle out of a constant number (4)of these power-of-two rectangles, which will give us the constant query time.

Representation: we represent the sparse table with a 4D array called S, with dimensions n by m by $\lg m$ by $\lg n$. In addition, we use a k-finding structure (described in Question 1) of length $max(m, n)$.

Preprocessing: Fill out the k-finding structure as in Question 1. This takes $O(max(m, n))$ time.

Next, we fill out the sparse table. For every location (i, j) we compute and store the AMQ for all rectangles with upper left corner (i, j) with width w and height h, for $w, h \in \{2^0, 2^1, ...2^k\}$ as long as the rectangles fit inside the original area. The maximum dimensions of the rectangle are $2^{\lg n}$ by $2^{\lg m}$ and there are $m * n$ potential upper-left corners so we have $O(mn \lg m \lg n)$ rectangles to compute.

We start by filling in $S[i][j][0][0] = A[i][j]$ for all i, j. This takes $O(mn)$ time.

We can fill in the rest using dynamic programming, in ascending order of area. We can compute all $1x2$ blocks by examining the two $1x1$ blocks that make them up; $2x2$ blocks by two of the four possible $1x2$ blocks that make them up; etc. This requires some tricky array indexing, but clearly every rectangle can be computed from a constant number (2) of smaller rectangles. Therefore each DP step is constant time, giving us a total runtime of $O(mn \lg m \lg n)$ necessary to compute and store the $O(mn \lg m \lg n)$ rectangles' AMQ values in the sparse table.

The total runtime of the preprocessing is $O(max(m, n) + mn + mn \lg m \lg n) = O(mn \lg m \lg n)$.

Querying: To answer $AMQ_A((i, j), (k, l))$:

We find the largest x such that $2^x \leq k - i + 1$ and the largest y such that $2^y \leq l - j + 1$.

We can overlap rectangles

$$B = ((i, j), (i + 2^x - 1, j + 2^y - 1)),$$
$$C = ((k - 2^x + 1, j), (k, j + 2^y - 1)),$$
$$D = ((k - 2^x + 1, l - 2^y + 1), (k, l)),$$
$$E = ((i, l - 2^y + 1), (i + 2^x - 1, l))$$

to form the original area $A = ((i, j), (k, l))$.

This is evident as an extension of the proof in 1 dimension shown in class. Each rectangle shares one corner with A, has width greater than half A's width and height greater than half A's height, so $B + C + D + E = A$.

We can look up any of those four rectangles in $O(1)$ time with an array access, e.g. $B = S[i][j][x][y]$, for a total query time of $O(1)$.

Question 3 (Hybrid RMQ Structures)**.**

i. Lemma: for any fixed $k \geq 1$, there is an RMQ data structure with time complexity $\langle O(n \lg^{(k)} n), O(1)\rangle$ called D_k.

Proof: by induction on k.

Base case: $k = 1$. A sparse table gives us time complexity $\langle O(n \lg n), O(1)\rangle$, proved in class.

Inductive step: given D_k, can we build D_{k+1}?

We use the hybrid approach with D_k as both our top and bottom RMQ structures, with a block size of log n.

Preprocessing time: $(p_1 = p_2 = O(n \lg^{(k)} n)))$

$$O(n + p_1(n/b) + (n/b)p_2(b))$$

We have $p_1(n/b) = (n/\lg n) \lg^{(k)}(n/\lg n) \leq (n/\lg n) \lg^{(k)} n$ (since $(n/\lg n) \leq n$ for sufficiently large n)

And then $(n/\lg n) \lg^{(k)} n \leq (n/\lg n) \lg n = n$ (since $\lg^{(k)} n \leq \lg n$) so $p_1(n/b) = O(n)$.

$$= O(n + n + (n/\lg n)b \lg^{(k)} b)$$

$$= O(2n + (n/\lg n) \lg n \lg^{(k)} \lg n)$$

$$= O(n \lg^{(k+1)} n)$$

Query time: $(q_1 = q_2 = O(1))$

$$O(q_1(n/b) + q_2(b))$$
$$= O(1)$$

So this hybrid approach is D_{k+1}, completing the induction.

ii. Query times increase because the constant factor increases. The asymptotic runtime is $O(1)$ because the runtime does not depend on n, only on k–the number of logs we're taking–which is not related to the size of the input. The source of the runtime increase is that each level of hybridization requires us to build two RMQ structures for the top and the bottom, which are hybrids themselves, and so on.

Question 4 (Implementation)**.** *Turned in electronically.*